REMARKS

This amendment is in response to the office action of May 4, 2007.

Regarding the 35 USC 112, sixth paragraph, language, it has been removed since applicant does not want that kind of a construction.

Regarding the statutory subject matter rejection of claim 33, it has been amended to overcome this rejection and withdrawal thereof is requested.

Regarding the novelty rejection, *Nishihara et al.* (US-6,026,427) discloses a method and apparatus which provide a condition variable that has a time-out capability for an operating system that does not have one. A condition variable allows concurrent programming and provides high level communication between threads. The condition variable allows threads to wait within a mutual exclusion (mutex) and allows broadcast signals to wake up waited threads. The condition variable is implemented by using counters to count the number of waited threads, the number of signals being sent to the waited threads, and semaphore, and means for mutually locking the thread to access the counters. (Abstract)

The condition variables allow for mutual exclusion of asynchronously interacting processors in self-directed distributed systems and provide a communication protocol that allow the systems to be used in time-critical applications and that minimizes wasted computing time due to idling processors. The apparatus of *Nishihara et al.* includes a system that controls access to a shared resource in a multi-processing computing environment. The method provides time-out capable condition variables, which are similar to semaphores, but are different from semaphores in that the condition variables allow threads to wait within a mutual exclusion and allow signals to be broadcast. The condition variables are implemented by using a semaphore and counters to count the number of waited threads and the number of threads signaling to the semaphores. (Col. 2, lines 9—23)

On Col. 3, lines 41—53 it reads: "*A condition variable is similar in some respects to a semaphore. Whereas a semaphore allows processes, or threads, to synchronize by controlling their access to data, a condition variable allows threads to synchronize on the value of the data. Cooperating threads wait until data reaches some particular state or until* *a particular event occurs. Thus, a condition variable is a synchronization object that allows a thread to become locked, until it is unlocked by some event. The unlocking can occur simultaneously, or as a result of either a time-out or some other thread performing a signaling operation on the condition variable. In use, condition variables are always associated with a mutual exclusion (mutex).*"

The condition variables are provided as modules each comprising a semaphore, a signal counter and a waited counter. The signal counter shows the number of signals issued by the thread. (Col. 3, lines 32—40)

On Col. 3, lines 53—60 it reads: "*A thread is a single sequential flow of control in a process. A thread may be currently processing or may be waited (i.e., its processing is suspended). A mutex is a synchronization object used to allow multiple threads to serialize their access to shared data. A mutex provides mutual exclusion such that a thread that has locked a mutex becomes the owner, and remains the owner, until the same thread unlocks the mutex.*"

According to description on col. 3, line 61—col. 4, line 2, waited threads can wait until a given event, such as a signal semaphore being sent to the thread or being broadcast to all threads, or the waited threads can wait an indefinite time. Alternately, waited threads can wait until receipt of the semaphore (signal or broadcast) or until the occurrence of a time-out. If a time-out wakes a waited thread before the semaphore is received, the semaphore counter shows 1 available count. This means that the next thread cannot wait correctly. The condition variable *signal count* is used in *Nishihara et al.* to overcome the problem relating to the error in the semaphore counter due to a time-out situation. When a thread has been woken up the value of the signal count is checked to see whether the reason to wake up the thread is a signal sent to the thread or a time-out.

On page 12, lines 11—17 of the subject application it reads: "*The activities included in the Application Session Profiles are implemented in the form of Activity Blocks which are distributed into Activity Block Containers ABC in an appropriate manner from the viewpoint of the capacity performance of the system, each of them containing Activity Blocks AB of one or more sessions. An individual Activity Block AB may be included in different sessions when processed at different times.*"

Further, in the last paragraph on page 12 of the subject application it is stated that the term session may be, for example, a simple list of program code comprising a single activity in the form of a single Activity Block, or a session may comprise a set of separate program modules or the like which can be arranged into one or more Activity Blocks. According to *Nishihara et al.*, a thread is a single sequential flow of control in a process. Therefore, thread and activity block are not analogous to each other.

On page 13 of the subject application it is disclosed that parallel ongoing application sessions comprise a set of successive and/or parallel activities, the respective Activity Blocks being located and executed in one or more Activity Block Containers, depending on the required parallelism. Activity Blocks are reusable high level modules of application sessions which do not, however, appear in a thread ready list of an underlying operating system. An Activity Block Container is a schedulable thread from the viewpoint of an underlying operating system.

The Examiner also argues that the condition variable module 45 of *Nishihara et al.* is analogous to the resource type specific resource handlers and the resource allocation manager of the subject application. Resource handlers are schedulable threads from the viewpoint of an underlying operating system. Condition variables are used to control the prosecution of the threads (lock, unlock). The condition variables are not controlling the operation of resources.

Further, neither the semaphore 50 nor the condition variable module 45 of *Nishihara et al.* are used to select the next application session and activity block to be executed on the basis of the resource allocation situation.

On the basis of above, *Nishihara et al.* is not teaching all the features of the independent claims of the subject application.

Withdrawal of the novelty rejection based on *Nishihara et al* is requested.

Regarding the novelty rejection based on *Alford et al.*, they disclose a portable thread environment comprising: an application programming interface configured to support multiple application program tasks, wherein each task is either a preemptive task comprised of preemptive threads or a cooperative task comprised of cooperative threads; host adaptation logic for communicatively interfacing said cooperative tasks, preemptive tasks, cooperative threads and preemptive threads with a host processing environment; a scheduler configured to determine an execution order of cooperative threads and preemptive threads based on each cooperative thread's and preemptive thread's priority levels. (Abstract)

Alford et al. solves a different problem than the subject application. *Alford et al.* deals with a portability of applications. According to paragraphs [0012] and [0013], *the tradition approach to porting applications is to write the application program in a "high-level language" that hopefully can be recompiled to generate machine code that can run within any of the prospective processors. ... A problem with the traditional porting method is that this method requires that at least some portion of the application program be rewritten. This is a potentially costly and error-prone process. Because there is a likelihood of introducing unintentional errors whenever the application program is altered, this method mandates that the application developer bare the additional expense of re-testing the application after the indicated changes are complete.*

According to the Summary of the Invention of *Alford et al.* there is provided a portable thread environment comprising: an application programming interface configured to support multiple application program tasks, wherein each task is either a preemptive task comprised of preemptive threads or a cooperative task comprised of cooperative threads; host adaptation logic for communicatively interfacing said cooperative tasks, preemptive tasks, cooperative threads and preemptive threads with a host processing environment; a scheduler configured to determine an execution order of cooperative threads and preemptive threads based on each cooperative thread's and preemptive thread's priority levels and task membership.

In the operating system's point of view the Portable Thread Environment ("PTE") of *Alford et al.* is only one thread. The PTE contains the operations such as the scheduler needed for the execution of an application: "*When the host environment includes operating system services, the PTE is scheduled and executed as an operating system task with the PTE's application program(s) 340 contained therein. In other words, the application* *program(s) 340 and the operation system 310 are isolated from one-another by the host adaptation layer 320 and the PTE interface 330.*" (Paragraph [0042])

In the paragraph [0038] it is stated that threads are executed by the PTE scheduler 220 in a sequence determined by scheduling variables such as, for example, the message order in the PTE scheduling queue 215, and/or the priority of messages stored in the queue 215. This is not the same than what the resource allocation manager of the subject application does. The scheduling queue is not taking into account resource allocation situation.

The Examiner also argues that the message sources 205, 206 in Fig. 2 correspond with the protocol of the subject application *connecting the Resource Handlers, Resource Allocation Manager, Application Session Management and Scheduling means and executing means, to control the execution order and to implement the transfer of information between said Resource Handlers, Resource Allocation Manager, Application Session Management and Scheduling means, and executing means*. However, the message sources only describe internal and external sources of messages. These messages may be used to form the scheduling queue 215 in a form of a list: "*The scheduling queue 215 in one embodiment is a list formed as messages are received from internal sources 206 such as running threads and from external sources 205 with which the application interacts.*" (paragraph [0038])

As to the claim 5, the Examiner argues that *Alford et al.* teaches all the features of claim 5 and refers to the PTE. However, as shown above, the paragraph [0042] teaches that the application programs and the operating system are isolated from one-another by the host adaptation layer and the PTE interface. *Alford et al.* does not disclose, for example, that PTE comprises a session control protocol comprised of application-independent control messages and rules on their use. All the messaging shown in figure 2 relates to the same application within the PTE.

The Examiner has only provided quite general arguments without detailed indication to relevant passages to the cited documents. Therefore, and on the basis of the above analysis, the *Alford et al* reference also fails to teach all the features of the independent claims and withdrawal of the novelty rejection is requested.

Without prejudice, and not to overcome the novelty rejections, but merely to make the claims more focused on the invention, some amendments have been made to improve the clarity and make the claims even more different from the prior art.

Basically, the independent claim 1 has been amended by including the subject matter of claim 5 into it. The other independent claims have been similarly amended.

It is believed that this reply is timely but if we have overlooked the need for an extension of time, the Commissioner is authorized to consider this paper as a petition for the appropriate extension period and to deduct the appropriate fee from our Deposit Account No. 23-0442 and to consider this paper as a petition therefor. We have enclosed a fee for the extra independent claim 34 but if there is no such fee enclosed or it is an incorrect fee, please credit or debit our Deposit Account No. 23-0442 to satisfy the fee due. Likewise if we have overlooked any other fee that would lead to abandonment for failure to pay same, please deduct it from our Deposit Account No. 23-0442.

The objections and rejections of the Office Action of May 4, 2007, having been obviated by amendment or shown to be inapplicable withdrawal thereof is requested and passage of claims 1-33 to issue is earnestly solicited.

Respectfully submitted,

Francis Maguire

Francis J. Maguire
Attorney for the Applicant
Registration No. 31,391

FJM/mo
Ware, Fressola, Van Der Sluys & Adolphson LLP
755 Main Street, P.O. Box 224
Monroe, CT 06468
(203) 261-1234